VLSI CAD Flow: Logic Synthesis, Placement and Routing

6.375 Lecture 5

Guest Lecture by Srini Devadas
RTL Design Flow

- HDL
  - RTL Synthesis
    - netlist
  - logic optimization
    - netlist
  - physical design
    - layout

Library/module generators

manual design

- a
- b
- s
- q
- d
- clk

- a
- b
- s
- q
- d
- clk

- a
- b
- s
- q
- d
- clk
Two-Level Logic Minimization

Can realize an arbitrary logic function in sum-of-products or two-level form

\[ F_1 = \overline{A} \overline{B} + \overline{A} AB D + \overline{A} B \overline{C} \overline{D} + A B C \overline{D} + A \overline{B} + A B D \]

\[ F_1 = \overline{B} + D + \overline{A} \overline{C} + A C \]

Of great interest to find a minimum sum-of-products representation

– Solved problem even for functions with 100’s of inputs (variants of Quine-McCluskey)
Two-Level versus Multilevel

2-Level:

\[ f_1 = AB + AC + AD \]
\[ f_2 = \overline{AB} + \overline{AC} + \overline{AE} \]

6 product terms which cannot be shared.
24 transistors in static CMOS

Multi-level:

Note that \( B + C \) is a common term in \( f_1 \) and \( f_2 \)

\[ K = B + C \]
\[ f_1 = AK + AD \]
\[ f_2 = \overline{AK} + \overline{AE} \]

3 Levels
20 transistors in static CMOS
not counting inverters
Technologies

“Closed book”: gate-array, standard-cell

“Open book”: CMOS Domino, complex gate static CMOS

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

Factoring, Commonality Extraction

TECH-DEPENDENT OPTIMIZATION (MAPPING, TIMING)

LIBRARY

OPTIMIZED LOGIC NETWORK
Tech.-Independent Optimization

Involves:
- Minimizing two-level logic functions.
- Finding common subexpressions.
- Substituting one expression into another.
- Factoring single functions.

Factored versus Disjunctive forms

\[ f = ac + ad + bc + bd + a\bar{e} \]
sum-of-products or disjunctive form

\[ f = (a + b)(c + d) + a\bar{e} \]
factored form

multi-level or complex gate
Optimizations

\[ F = \begin{cases} 
  f_1 &= AB + AC + AD + AE + \overline{ABCDE} \\
  f_2 &= \overline{AB} + \overline{AC} + \overline{AD} + \overline{AF} + \overline{ABCDF} 
\end{cases} \]

Factor \( F \)

\[ F = \begin{cases} 
  f_1 &= A(B + C + D + E) + \overline{ABCDE} \\
  f_2 &= \overline{A}(B + C + D + F) + \overline{ABCDF} 
\end{cases} \]

Extract common expression

\[ G = \begin{cases} 
  g_1 &= B + C + D \\
  f_1 &= A(g_1 + E) + \overline{A}Eg_1 \\
  f_2 &= \overline{A}(g_1 + F) + \overline{AF}g_1 
\end{cases} \]
What Does “Best” Mean?

- Transistor count → AREA
- Number of circuits → POWER
- Number of levels → DELAY (Speed)

Need quick estimators of area, delay and power which are also accurate
Algebraic vs. Boolean Methods

Algebraic techniques view equations as polynomials and attempt to factor equations or “divide” them

Do not exploit Boolean identities e.g., \( a \overline{a} = 0 \)

In algebraic substitution (or division) if a function \( f = f(a, b, c) \) is divided by \( g = g(a, b) \), \( a \) and \( b \) will not appear in \( f/g \)

Algebraic division: \( O(n \log n) \) time

Boolean division: 2-level minimization required
Comparison

\[ f = a\overline{b} + a\overline{c} + b\overline{a} + b\overline{c} + c\overline{a} + c\overline{b} \]

Algebraic factorization procedures

\[ f = a(\overline{b} + \overline{c}) + \overline{a}(b + c) + b\overline{c} + c\overline{b} \]

Boolean factorization produces

\[ f = (a + b + c)(\overline{a} + \overline{b} + \overline{c}) \]

\[ l = (b\overline{f} + \overline{b}f) (a + e) + \overline{a}\overline{e}(\overline{b}f + bf) \]

\[ r = (b\overline{f} + \overline{b}f) (\overline{a} + \overline{e}) + ae(\overline{b}f + bf) \]

Algebraic substitution of \( l \) into \( r \) fails

Boolean substitution

\[ r = a(\overline{e}\overline{l} + el) + \overline{a}(\overline{e}l + e\overline{l}) \]

\[ l = a(er + \overline{e}r) + \overline{a}(er + e\overline{r}) \]
Strong (or Boolean) Division

Given a function $f$ to be strong divided by $g$

Add an extra input to $f$ corresponding to $g$, namely $G$ and obtain function $h$ as follows

$$h_{DC} = G\overline{g} + \overline{G}g$$

$$h_{ON} = f_{ON} - h_{DC}$$

Minimize $h$ using two-level minimizer
Strong Division Example

\[ f = \bar{a} \bar{b} c + \bar{a} b \bar{c} + a \bar{b} \bar{c} + a b c \]

\[ g = a \bar{b} + \bar{a} b \]

Minimization gives

\[ h = \overline{G_c} + G \overline{c} \]

**Function \( h \)**

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(x)</td>
<td>1</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>(x)</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>(x)</td>
<td>1</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(x)</td>
<td>(x)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Weak (or Algebraic) Division

Definition: support of $f$ as $\text{sup}(f) = \{ \text{set of all variables } v \text{ that occur in } f \text{ as } v \text{ or } \overline{v} \}$

Example: $f = A \overline{B} + C$

$\text{sup}(f) = \{ A, B, C \}$

Definition: we say that $f$ is orthogonal to $g$, $f \perp g$, if $\text{sup}(f) \cap \text{sup}(g) = \phi$

Example: $f = A + B$ \hspace{2cm} g = C + D

$\therefore f \perp g \text{ since } \{A, B\} \cap \{C, D\} = \phi$
We say that \( g \) divides \( f \) weakly if there exist \( h, r \) such that \( f = gh + r \) where \( h \neq \phi \) and \( g \perp h \)

Example:

\[
\begin{align*}
f &= ab + ac + d \\
g &= b + c \\
f &= a(b + c) + d & h = a & r = d
\end{align*}
\]

We say that \( g \) divides \( f \) evenly if \( r = \phi \)

The quotient \( f/g \) is the largest \( h \) such that \( f = gh + r \) i.e., \( f = (f/g)g + r \)
Weak Division Example

\[ f = abc + abde + abh + bcd \]
\[ g = c + de + h \]

**Theorem:** \[ f/g = f/c \cap f/de \cap f/h \]

- \[ f/c = ab + bd \]
- \[ f/de = ab \]
- \[ f/h = ab \]

\[ f/g = (ab + bd) \cap ab \cap ab = ab \]
\[ f = ab(c + de + h) + bcd \]

**Time complexity:** \[ O(\mid f \mid \mid g \mid) \]
How to Find Good Divisors?

$64K$ question

Strong division: Use existing nodes in the multilevel network to simplify other nodes

Weak division: Generate good algebraic divisors using algorithms based on “kernels” of an algebraic expression
Tech.-Dependent Optimization

OPTIMIZED LOGIC EQUATIONS

LIBRARY

TECHNOLOGY MAPPING

TIMING CONSTRAINTS

GATE NETLIST

Area, delay and power dissipation cost functions
“Closed Book” Technologies

A standard cell technology or library is typically restricted to a few tens of gates e.g., MSU library: 31 cells

Gates may be NAND, NOR, NOT, AOIs.
Mapping via DAG Covering

Represent network in canonical form
⇒ subject DAG

Represent each library gate with canonical forms for the logic function
⇒ primitive DAGs

Each primitive DAG has a cost

Goal: Find a minimum cost covering of the subject DAG by the primitive DAGs

Canonical form: 2-input NAND gates and inverters
Sample Library

INVERTER 2

NAND2 3

NAND3 4

NAND4 5
Sample Library - 2

AOI21 4

AOI22 5
Trivial Covering

subject DAG

7  NAND2  =  21
5  INV    =  10

31
Covering #1

2 INV = 4
2 NAND2 = 6
1 NAND3 = 4
1 NAND4 = 5

19
Covering #2

1 INV = 2
1 NAND2 = 3
2 NAND3 = 8
1 AOI21 = 4

17
Sound Algorithmic approach
NP-hard optimization problem

Tree covering heuristic: If subject and primitive DAGs are trees, efficient algorithm can find optimum cover in linear time
⇒ dynamic programming formulation
Partitioning a Graph
Resulting Trees

Break at multiple fanout points
Dynamic Programming

Principle of optimality: Optimal cover for a tree consists of a match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match.

Best cover for this match uses best covers for x, y, z

Best cover for this match uses best covers for p, z
Optimum Tree Covering

INV
11 + 2 = 13

NAND2
2 + 6 + 3 = 11

INV
2

NAND2
3 + 3 = 6

NAND2

AOI21
4 + 3 = 7

NAND2
3
RTL Design Flow

Library/module generators

HDL

RTL Synthesis

netlist

logic optimization

netlist

physical design

layout

manual design
Physical Design: Overall Conceptual Flow

Input

Floorplanning

- Read Netlist

Placement

- Floorplanning
- Initial Placement
- Routing Region Definition
- Global Routing
- Cost Estimation
- Placement Improvement

Routing

- Routing Region Ordering
- Detailed Routing
- Cost Estimation
- Routing Improvement

Output

- Compaction/clean-up
- Write Layout Database
Results of Placement

A bad placement

A good placement

What’s good about a good placement?
What’s bad about a bad placement?
Results of Placement

Bad placement causes routing congestion resulting in:

- Increases in circuit area (cost) and wiring
- Longer wires → more capacitance
  - Longer delay
  - Higher dynamic power dissipation

Good placement

- Circuit area (cost) and wiring decreases
- Shorter wires → less capacitance
  - Shorter delay
  - Less dynamic power dissipation
**Gordian Placement Flow**

Data flow in the placement procedure GORDIAN

**Complexity**
- space: $O(m)$
- time: $Q(m^{1.5} \log^2 m)$

**Final placement**
- standard cell
- macro-cell & SOG
Gordian: A Quadratic Placement Approach

- Global optimization: solves a sequence of quadratic programming problems
- Partitioning: enforces the non-overlap constraints
Intuitive formulation

Given a series of points $x_1, x_2, x_3, \ldots x_n$
and a connectivity matrix $C$ describing the connections
between them

(If $c_{ij} = 1$ there is a connection between $x_i$ and $x_j$)

Find a location for each $x_j$ that minimizes the total sum of
all spring tensions between each pair $<x_i, x_j>$

\[ \text{x}_i \quad \text{x}_j \]

Problem has an obvious (trivial) solution – what is it?
**Improving the intuitive formulation**

To avoid the trivial solution add constraints: \(Hx=b\)

- These may be very natural - e.g. endpoints (pads)

\[
x_1 \quad \ldots \quad x_n
\]

To integrate the notion of "critical nets"

- Add *weights* \(w_{ij}\) to nets

\[
i \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad j
\]

\(w_{ij}\) - some springs have more tension should pull associated vertices closer
Modeling the Net’s Wire Length

The length $L_v$ of a net $v$ is measured by the squared distances from its points to the net’s center

$$L_v = \sum_{u \leftarrow M_v} [(x_{uv} - x_v)^2 + (y_{uv} - y_v)^2]$$

$$\begin{align*}
(x_{uv} &= x_u + \xi_{uv} ; \quad y_{uv} = y_u + y_{vu})
\end{align*}$$
Cost = \((x_1 - 100)^2 + (x_1 - x_2)^2 + (x_2 - 200)^2\)

\[ \frac{\partial}{\partial x_1} \text{Cost} = 2(x_1 - 100) + 2(x_1 - x_2) \]

\[ \frac{\partial}{\partial x_2} \text{Cost} = -2(x_1 - x_2) + 2(x_2 - 200) \]

Setting the partial derivatives = 0, we solve for the minimum Cost:

\[ A\mathbf{x} + \mathbf{B} = 0 \]

\[
\begin{bmatrix}
4 & -2 \\
-2 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
+
\begin{bmatrix}
-200 \\
-400
\end{bmatrix} = 0
\]

\[
\begin{bmatrix}
2 & -1 \\
-1 & 2
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
+
\begin{bmatrix}
-100 \\
-200
\end{bmatrix} = 0
\]

\[ x_1 = \frac{400}{3} \quad x_2 = \frac{500}{3} \]

Toy Example:
Quadratic Optimization Problem

\[ \begin{align*}
A & \quad B & \quad C & \quad D & \quad E & \quad F & \quad G \\
\vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots \\
\rho & \quad * & \quad * & \quad 0 & \quad 0 & \quad 0 & \quad 0 \\
\rho' & \quad 0 & \quad 0 & \quad 0 & \quad * & \quad * & \quad * \\
\vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots & \quad \vdots \\
A(l) &= \begin{pmatrix}
\rho & \\
\rho' & \\
\end{pmatrix} \\
A &= \begin{pmatrix}
A(l) \\
\end{pmatrix} \\
\end{align*} \]

- Linearly constrained quadratic programming problem

\[ \begin{align*}
\min_{x \in \mathbb{R}^m} \{ \Phi(x) = x^T C x + d^T x \} \\
\text{s.t. } A(l)x = u(l) \\
\end{align*} \]

Accounts for fixed modules
Wire-length for movable modules
Center-of-gravity constraints

Problem is computationally tractable, and well behaved
Commercial solvers available: mostek
Global Optimization Using Quadratic Placement

Quadratic placement clumps cells in center

Partitioning divides cells into two regions
  * Placement region is also divided into two regions

New center-of-gravity constraints are added to the constraint matrix to be used on the next level of global optimization
  * Global connectivity is still conserved
Setting up Global Optimization

Fig. 1. Data flow in the placement procedure GORDIAN.
Layout After Global Optimization

A. Kahng
Fig. 1. Data flow in the placement procedure GORDIAN.
Partitioning

In GORDIAN, partitioning is used to constrain the movement of modules rather than reduce problem size.

By performing partitioning, we can iteratively impose a new set of constraints on the global optimization problem:

- Assign modules to a particular block

Partitioning is determined by:

- Results of global placement – initial starting point
  - Spatial (x,y) distribution of modules
- Partitioning cost
  - Want a min-cut partition
Layout after Min-cut

Now global placement problem will be solved again with two additional center_of_gravity constraints
Adding Positioning Constraints

• Partitioning gives us two new “center of gravity” constraints

• Simply update constraint matrix

• Still a single global optimization problem

• Partitioning is not “absolute”
  • modules can migrate back during optimization
  • may need to re-partition

\[
\begin{align*}
\mathbf{A}^{(l)} &= \frac{\varrho}{\varrho'}
\begin{bmatrix}
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
* & * & * & 0 & 0 & 0 & \ddots & \ddots \\
0 & 0 & 0 & * & * & * & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}
\end{align*}
\]

Fig. 4. The constraints for global placement.
Fig. 1. Data flow in the placement procedure GORDIAN.
First Iteration
Second Iteration
Third Iteration
Fourth Iteration
Final Placement

Fig. 1. Data flow in the placement procedure GORDIAN.
Earlier steps have broken down the problem into a manageable number of objects.

Two approaches:

- Final placement for standard cells/gate array – row assignment
- Final placement for large, irregularly sized macro-blocks – slicing – won’t talk about this
This process continues until there are only a few cells in each group (≈ 6)

Assign cells in each group close together in the same row or nearly in adjacent rows

group: smallest partition

A. E. Dunlop, B. W. Kernighan,
Partitioning of circuit into 32 groups. Each group is either assigned to a single row or divided into 2 rows.
Standard Cell Layout
Another Series of Gordian

(a) Global placement with 1 region  (b) Global placement with 4 region  (c) Final placements

D. Pan – U of Texas
Physical Design Flow

Input

Floorplanning

Read Netlist

Floorplanning

Initial Placement

Routing Region Definition

Global Routing

Cost Estimation

Routing Region Ordering

Detailed Routing

Cost Estimation

Placement

Compaction/clean-up

Routing

Detailed Routing

Cost Estimation

Output

Write Layout Database

Placement Improvement

Routing Improvement

 Courtesy K. Keutzer et al. UCB
Kahng/Keutzer/Newton
Imagine …

- You have to plan transportation (i.e. roads and highways) for a new city the size of Chicago.
- Many dwellings need direct roads that can’t be used by anyone else.
- You can affect the layout of houses and neighborhoods but the architects and planners will complain.
- And … you’re told that the time along any path can’t be longer than a fixed amount.
- What are some of your considerations?
What are some of your considerations?

- How many levels do my roads need to go? Remember: Higher is more expensive.
- How do I avoid congestion?
- What basic structure do I want for my roads?
  - Manhattan?
  - Chicago?
  - Boston?

- Automated route tools have to solve problems of comparable complexity on every leading edge chip
Routing Applications

- Cell-based
- Mixed Cell and Block
- Block-based
Routing Algorithms

Hard to tackle high-level issues like congestion and wire-planning and low level details of pin-connection at the same time

- **Global routing**
  - Identify routing resources to be used
  - Identify layers (and tracks) to be used
  - Assign particular nets to these resources
  - Also used in floorplanning and placement

- **Detail routing**
  - Actually define pin-to-pin connections
  - Must understand most or all design rules
  - May use a compactor to optimize result
  - Necessary in all applications
Basic Rules of Routing - 1

- Wiring/routing performed in layers – 5-9 (-11), typically only in “Manhattan” N/S E/W directions
  - E.g. layer 1 – N/S
  - Layer 2 – E/W

- A segment cannot cross another segment on the same wiring layer

- Wire segments *can* cross wires on other layers

- Power and ground may have their own layers

Photo courtesy:
Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Kahng/Keutzer/Newton
Basic Rules of Routing – Part 2

- Routing can be on a fixed grid –

- Case 1: Detailed routing only in channels
  - Wiring can only go over a row of cells when there is a free track – can be inserted with a “feedthrough”
  - Design may use of metal-1, metal-2
  - Cells *must* bring signals (i.e. inputs, outputs) out to the channel through “ports” or “pins”
Basic Rules of Routing – Part 3

- Routing can be on a fixed or gridless (aka area routing)

- Case 1: Detailed routing over cells
  - Wiring can go over cells
  - Design of cells must try to minimize obstacles to routing – i.e. minimize use of metal-1, metal-2
  - Cells do not need to bring signals (i.e. inputs, outputs) out to the channel – the route will come to them
Taxonomy of VLSI Routers

- **Global**
  - Graph Search
  - Steiner
    - Iterative

- **Detailed**
  - Restricted
    - River
    - Switchbox
    - Channel
  - General Purpose
    - Maze
    - Line Probe
    - Line Expansion

- **Specialized**
  - Power & Ground
    - Clock

- **Routers**
  - Hierarchical
  - Greedy
  - Left-Edge

Courtesy K. Keutzer et al. UCB
Today’s high-perf logical/physical flow

1) optimize using estimated or extracted capacitances
2) re-place and re-route
3) if design fails to meet constraints due to poor estimation - repeat 1 + 2 -
Top-down problems in the flow

- Library
- netlist
- user constraints
- tech files
- logic optimization/timing verif
- placement
- routing
- layout
- delay model generator
- SDF cell/wire delays
- RC
- extraction

Initial capacitance estimates inaccurate
Inability to take top-down timing constraints
Inaccurate internal timing model
Iteration problems in the flow

- Updated capacitances cause significant changes in optimization.
- Limited-incremental capability.
- Resulting iteration may not bring closer to convergence.